Vietnam J Comput Sci (2016) 3:181–196
DOI 10.1007/s40595-016-0062-4

REGULAR PAPER

Similarity search for numerous patterns over multiple time series streams under dynamic time warping which supports data normalization

Bui Cong Giao[1] · **Duong Tuan Anh**[1]

Received: 19 December 2015 / Accepted: 24 February 2016 / Published online: 11 March 2016

Abstract A huge challenge in nowadays' data mining is similarity search in streaming time series under Dynamic Time Warping (DTW). In the similarity search, data normalization is a must to obtain accurate results. However, data normalization on the fly and the DTW calculation cost a great deal of computational time and memory space. In the paper, we present two methods, SUCR-DTW and ESUCR-DTW, which conduct similarity search for numerous prespecified patterns over multiple time-series streams under DTW supporting data normalization. These two methods utilize a combination of techniques to mitigate the aforementioned costs. The efficient methods inherit the cascading lower bounds introduced in UCR-DTW, a state-of-the-art method of similarity search in the static time series, to admissibly prune off unpromising subsequences. To be adaptive in the streaming setting, SUCR-DTW performs incremental updates on the envelopes of new-coming time-series subsequences and incremental data normalization on time-series data. However, like UCR-DTW, SUCR-DTW retrieves only similar subsequences that have the same length as the patterns. ESUCR-DTW, an extension of SUCR-DTW, can find similar subsequences whose lengths are different from those of the patterns. Furthermore, our proposed methods exploit multi-threading to have a fast response to high-speed time-series streams. The experimental results show that SUCR-DTW obtains the same precision as UCR-DTW and has lower wall clock time. Besides, the experimental results of SUCR-DTW and ESUCR-DTW reveal that the extended method has higher accuracy in spite of longer wall clock time. Also, the paper evaluates the influence of incremental z-score normalization and incremental min–max normalization on the obtained results.

Keywords Similarity search · Time series stream · Dynamic time warping · Data normalization

1 Introduction

A time-series stream is a sequence of data collected in a continuous manner as time progresses. In recent years, due to accelerated technology developments, there have been more and more applications related to data mining in streaming time series, ranging from monitoring of sensor networks [1] and environmental signals [2], to trading stocks online [3]. In such applications, similarity search for prespecified patterns in streaming time series is a critical subroutine, yet the time taken for the task is almost a hurdle, since time-series streams might transfer huge amount of data at steady high-speed rates. Hence, time-series streams are potentially unbounded in size within a short period and the system runs out of memory soon. Consequently, if a data point of time-series stream has been processed, it is quickly discarded and cannot be retrieved so that it yields to a new-coming one. To achieve real-time response, methods of similar search in streaming time series need to have one-pass scan and low computational time, yet available methods used to manage static time series are hardly able to satisfy the above requirements as they commonly need to scan time-series sequences many times and often have high computation cost. Therefore, according to Yang and Wu [4], high-speed data streams is the second ranking challenge among the ten top challenging problems in the present day' s data mining. Furthermore, Fu [5] has recently conducted a

✉ Bui Cong Giao
giao.bc@cb.sgu.edu.vn

Duong Tuan Anh
dtanh@cse.hcmut.edu.vn

[1] Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam

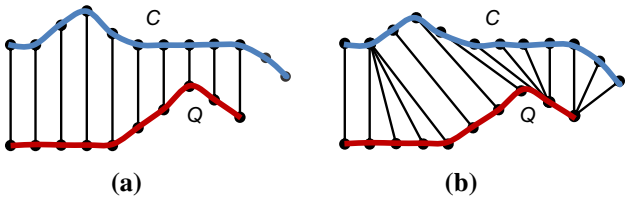

Fig. 1 Matching points of **a** the Euclidean distance and **b** the DTW distance

review on time-series data mining and claimed that mining on streaming time series is a fascinating research direction.

There are two popular distance metrics for similarity search in time-series data, the Euclidean distance and the Dynamic Time Warping (DTW) [6] distance. DTW, which originated in the speech community [7], is a robust distance measure, because the distance measure can find similar time-series sequences though they are misaligned and different in length, whereas the Euclidean metric can hardly do this. Therefore, results obtained from DTW are more accurate than those from the Euclidean metric. Figure 1 illustrates the difference between the two distance metrics in matching among points to calculate the distance between two time-series sequences C and Q. The applicability of DTW is not only in pattern discovery by similarity search over time-series data, but also in other time-series data-mining tasks such as classification [8] and clustering [9].

Data normalization is very important for similarity search over time-series data. Many researchers [10,11] reckoned that this preprocessing step is necessary to have meaningful results. For example, when two time series are analyzed concurrently, one collects rainfall, whereas another records humidity. Since these values are measured on different offsets, they cannot be compared meaningfully. Another example is given in Fig. 2 to justify the reason that data normalization is required for similarity search over time-series data. In Fig. 2b, the time-series sequence is a segment extracted from a real EEG data set [12]. We define a time-series pattern for query as in Fig. 2a, so as to retrieve similar subsequences in the time-series sequence. The result is that only one similar subsequence is found as shown in Fig. 2b and two other similar subsequences are missed due to shifting.

DTW is not only used for static time series but also for streaming time series, yet some accelerating techniques for the DTW calculation in similarity search over time-series streams work only on the un-normalized data (e.g. [13,14]); consequently, the obtained results are not accurate. Recently, Rakthanmanon et al. [11] have introduced UCR-DTW, a method of similarity search for patterns, which are prespecified time-series sequences, in static time series under DTW. The authors paid due attention to data normalization prior to any computation of the DTW distance. The experimental results of UCR-DTW reveal that the method has low computational time and high accuracy. The method, however, works only on static time series and requires two sequences of the same length while computing the DTW distance, so it leaves many things open in similarity search over time-series streams.

Motivated by the above observation, in this paper we will present two methods, SUCR-DTW and ESUCR-DTW, of similarity search for prespecified patterns in streaming time series under DTW, which support data normalization. We introduced SUCR-DTW in [15], which is a modification of UCR-DTW to be adaptive in the streaming context. In the work, we will describe SUCR-DTW in more detail and introduce ESUCR-DTW for the first time. ESUCR-DTW is an extension of SUCR-DTW for retrievals of similar subsequences whose lengths are likely to be different from those of the corresponding patterns. Besides, in ESUCR-DTW, the lengths of expectative similar subsequences can be prespecified within a valid domain for each pattern. The two methods can be used with any type of data normalization such as z-score normalization and min–max normalization provided that these data normalization types can be incrementally calculated, so as to mitigate the high computational time due to the course of data normalization in the streaming setting. More specifically, these two methods can deal with an important scenario in streaming applications where incoming data are from multiple concurrent time-series streams at high-speed rates, and there are numerous prespecified patterns for query.

As regards technical aspect, both of the proposed methods have salient characteristics as follows.

- Applicability of *multi-threading* for similarity search over multiple time-series streams.

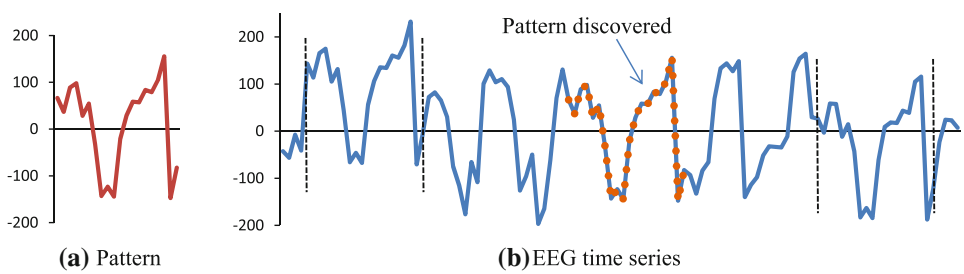

Fig. 2 An example illustrates why data normalization is required for similarity search over time-series data

- Incremental update on the envelopes of new-coming time-series subsequences so that these envelopes can be immediately used in a lower bounding function.

We conducted a large number of experiments to evaluate the efficacy of SUCR-DTW and ESUCR-DTW. Firstly, SUCR-DTW is compared with UCR-DTW in terms of precession and wall clock time. Next, ESUCR-DTW is compared with SUCR-DTW in terms of accuracy and wall clock time. Then, for the first time, the results obtained by ESUCR-DTW using incremental z-score normalization are compared with those done by ESUCR-DTW using incremental min–max normalization. Finally yet importantly, ESUCR-DTW is compared with SPRING, a well-known method of similarity search in streaming time series, combined with incremental min–max normalization in terms of wall clock time and the quality of similar subsequences.

The rest of paper is organized as follows. Section 2 describes DTW, techniques to speedup DTW, data normalization, and typical tasks of similarity search in streaming time series. Section 3 reviews related work. Afterwards, Sect. 4 describes our two proposed methods. Section 5 goes into the experimental evaluation, and Sect. 6 gives conclusions and future work.

2 Background

2.1 Dynamic Time Warping

This nonlinear distance measure allows time-series sequences to be stretched along the time axis to minimize the distance between the sequences. The DTW distance is calculated by dynamic programming as follows. Consider two time-series sequences $C = c_1, c_2, \ldots, c_m$ and $Q = q_1, q_2, \ldots, q_n$. The DTW distance between C and Q is defined as:

$$
\begin{aligned}
&DTW(C, Q) = f(m, n)\\
&f(m, n) = d(c_i, q_j) + \min \begin{cases} f(i, j-1) \\ f(i-1, j) \\ f(i-1, j-1) \end{cases} \qquad (1)\\
&f(0, 0) = 0, \quad f(i, 0) = f(0, j) = \infty\\
&(1 \le i \le m, 1 \le j \le n)
\end{aligned}
$$

where $d(c_i, q_j) = (c_i - q_j)^2$ is the Euclidean distance between two numerical values, c_i and q_j. Notice that any choice (e.g. $d(c_i, q_j) = |c_i - q_j|$) would be fine. Our proposed methods are completely independent of such choices. To align C and Q using DTW, an n-by-m accumulated cost matrix whose (ith, jth) cell contains the value of $f(c_i, q_j)$ is constructed. An optimal warping path P is a sequence of continuous cells in the matrix, which defines a mapping between C and Q such that $f(m, n)$ is minimum.

The calculation of the DTW distance can be expressed by a simpler way. Matching points of C and Q as in Fig. 1b creates an optimal warping path P as in Fig. 3a. Let the kth element of P be $p_k = (i, j)_k$. We have $P = p_1, p_2, \ldots, p_k, \ldots, p_K$, where $\max(m, n) \le K \le m + n - 1$. The DTW distance between C and Q is a cumulative addition along P, which minimizes the warping cost as follows:

$$
DTW(C, Q) = \sqrt{\sum_{k=1}^{K} d(p_k)}. \qquad (2)
$$

Because DTW uses a dynamic programming algorithm whose time and space complexity are $\mathcal{O}(mn)$, the distance

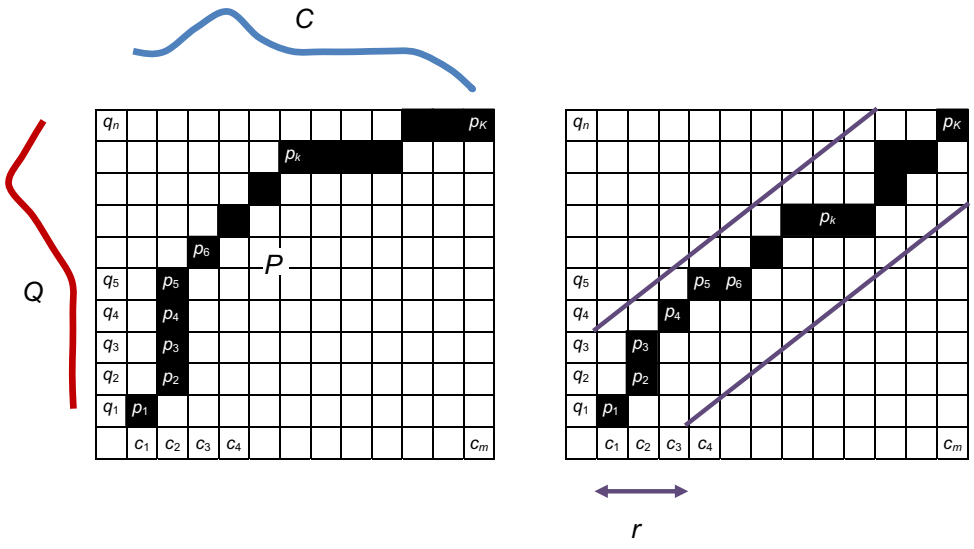

Fig. 3 **a** To align C and Q, a warping path P, shown with *solid squares*, is constructed. **b** The Sakoe–Chiba band with a width r is used as a global constraint to limit the scope of P

measure is almost very slow, especially for long time-series sequences. For that reason, there have been many incessant researches to speed up DTW, since Berndt and Clifford introduced the distance metric in 1994 [6].

2.2 Techniques to speedup Dynamic Time Warping

The techniques to speedup DTW often fall into three categories:

- *Constraints* The technique aims to limit the number of cells evaluated in the accumulated cost matrix. Figure 3b depicts a Sakoe–Chiba band [16] that prevents pathological warping paths, where a data point in one time-series sequence matches too many data points of another as in Fig. 1b. The Sakoe–Chiba band constrains a warping window in the area defined by two lines parallel to the diagonal. Keogh and Ratanamahatana [10] showed that restricting the size of the warping windows not only speedups computation, because only a part of the accumulated cost matrix needs computing, but also tightens the lower bounding property.
  The Sakoe–Chiba band works well in domains where an optimal warping path is expected to be close to the diagonal of the accumulated cost matrix. The constraint works poorly if time series are of events that start and stop at extremely different times because the warping path can stray very far from a linear warping path and nearly the entire matrix must be evaluated to find an optimal warping path.
- *Lower bounding* The technique uses cheap-to-compute lower bounding functions to reduce the number of times computing the DTW distance for finding the time-series sequence that is nearly similar to a given time-series pattern.
  Let F be a function of dimension reduction or feature extraction of a time-series sequence. A lower bounding function d_F is of the lower bounding property as follows.

$$d_F(F(C), F(Q)) \leq DTW(C, Q). \tag{3}$$

  The efficiency of d_F is evaluated in terms of time complexity and pruning power. The pruning power of d_F is competence for early detection of unpromising sequences so as not to use the naive DTW calculation on these sequences in the post-processing phase. Let g be the number of unpromising sequences which d_F identifies, and G be the total number of sequences which are performed in a similarity search. The pruning power of d_F is

$$\frac{100 \times g}{G}\ \%. \tag{4}$$

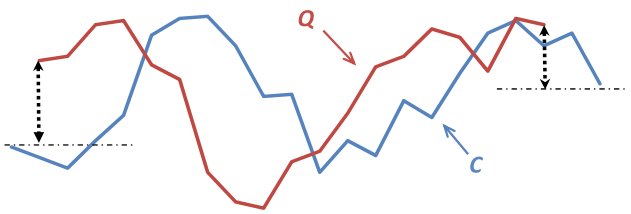

Fig. 4 LB_Kim on Q and C, which are normalized

In this work, we applied two efficient lower bounding functions that will be briefly described as follows. Firstly, the lower bounding measure proposed by Kim et al. [17] (hereafter, referred to as LB_Kim), uses the four-tuples features from each sequence. The features are the first and last data points of the sequence, together with the maximum and minimum values. However, Rakthanmanon et al. [11] believed that as time-series sequences are normalized, the distance values of these extra two-tuples (the maximum and minimum vectors) tend to be very small, so we may ignore them. As a result, the computation complexity of LB_Kim reduces from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. Figure 4 depicts an illustration of LB_Kim using the first points and last ones of Q and C.

Secondly, Keogh and Ratanamahatana [10] introduced another lower bounding technique (referred to as LB_Keogh). The technique exploits the fact that most DTW applications use global path constraints while comparing two time-series sequences, that is, i and j in $p_k = (i, j)_k$ are constrained to $j - r \leq i \leq j + r$, where r is independent of i in case of the Sakoe–Chiba band. Using the fact, two time series U and L (for upper and lower bounds) are constructed, such that they define an envelope that Q must lie in, regardless of how much it is skewed under all possible warping paths that are allowed under the global path constraint. Time-series data points of U and L are

$$\begin{aligned} u_i &= \max(q_{i-r} : q_{i+r}) \\ l_i &= \min(q_{i-r} : q_{i+r}). \end{aligned} \tag{5}$$

Figure 5 shows the computation of LB_Keogh on Q and C with U and L of Q. The lower bounding function computes the sum of the distances of the data points of C beyond the envelope of Q. The authors [10] reckoned two time-series sequences of the same length and constrained amount of warping produces no false dismissals. Thus, DTW has become a very powerful tool in time-series data mining since then. Having Q and C of same length, and U and L of Q, LB_Keogh is defined as follows:

$$LB\_Keogh(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (c_i - u_i)^2 & \text{if } c_i > u_i \\ (l_i - c_i)^2 & \text{if } c_i < l_i \\ 0 & \text{otherwise.} \end{cases}} \tag{6}$$

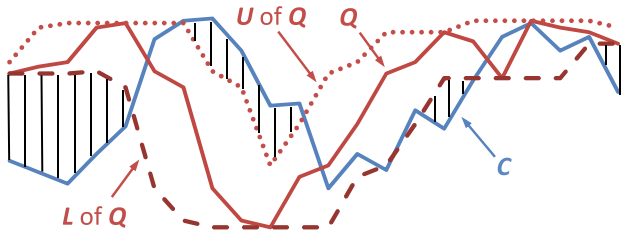

Fig. 5 LB_Keogh on Q and C whose length is n, so the computation complexity is $\mathcal{O}(n)$

- *Early abandoning* The technique is based on a comparison of distances with a threshold ε. Similarity search over two time-series sequences C and Q needs to check $f(i, j)$ in the formula (1), such that $f(i, j) \leq \varepsilon$. If the value of f exceeds ε, then C is not similar to Q and the course to compute $DTW(C, Q)$ is stopped immediately. Some works using the technique to accelerate the DTW calculation are [11,18]. Notice that early abandoning with a threshold ε can also be used in the calculation of the Euclidean distance. Moreover, the LB_Keogh lower bound can be used for early abandoning of the DTW calculation as follows. While the classical DTW calculation is being incrementally computed from left to right of two sequences Q and C (e.g. from 1 to k), if the partial DTW accumulation with the LB_Keogh contribution from $k+1$ to n exceeds ε, then the naive DTW calculation is aborted right away, since the sum of

$$DTW(Q_{1:k}, C_{1:k}) + LB\_Keogh(Q_{k+1:n}, C_{k+1:n})$$

is a lower bound of $DTW(Q_{1:n}, C_{1:n})$. Hence, on the occasion of the calculation of $LB\_Keogh(Q, C)$, an array of cumulative bounds is got from the lower bounding function. The kth element of the array of cumulative bounds is $LB\_Keogh(Q_{k:n}, C_{k:n})$.

2.3 Data normalization

Data normalization makes the results of data-mining tasks more accurate. Two common ways to normalize time-series data are min–max and z-score. Let X denote a time-series sequence, $X = x_1, x_2, \ldots, x_n$.

Min–max normalization maps a value x of X to x_{norm} by computing

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{7}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and the maximum values of time-series X. $x_{\min}$ and $x_{\max}$ are referred to as min–max coefficients.

Z-score normalization maps a value x of X to x_{norm} by computing

$$x_{\text{norm}} = \frac{x - \mu}{\sigma} \tag{8}$$

$$\text{with} \quad \mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{9}$$

$$\text{and} \quad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \mu^2. \tag{10}$$

μ and σ are referred to as z-score coefficients.

Z-score normalization is often used in data-mining tasks on time-series data, since normalized time-series sequences follow the shape of original time-series ones more closely; however, z-score normalization does not make sure that normalized time-series sequences are of the same amplitude. For instance, in image processing, pixel intensities have to be normalized to fit within a range from 0 to 255 for the RGB colour range. Also, a typical algorithm of neural network requires data on a 0–1 scale, so min–max normalization can be utilized to get values within the range. Furthermore, min–max normalization is of low computational time. Thus, there have been many recent time-series applications (e.g. [19,20]) using min–max normalization.

2.4 Typical tasks of similarity search in streaming time series

A time-series stream X is a discrete, semi-infinite time-series sequence of real numbers $x_1, x_2, \ldots, x_n$, where x_n is the most recent value. In other words, X is a univariate time series, which is evolving with an increase of n after each time tick. Let $X[x_s : x_e]$ be the subsequence starting from time tick s, and ending at e; and $NX[nx_s : nx_e]$ be the normalized subsequence of $X[x_s : x_e]$. Let $Y[y_0 : y_{m-1}]$ be a time-series pattern of length m, and $NY[ny_0 : ny_{m-1}]$ be the normalized sequence of Y. Three typical tasks of similarity search for Y over X until the most recent time tick n are:

- *Best-so-far search*: Finding such an NX that is most similar to NY. That means $DTW(NX, NY)$ is smallest. The smallest value, which is recorded until time tick n, is the *best-so-far* value, and X is the *best-so-far* subsequence of Y.
- *Range search*: Given a threshold ε, finding any NX such that $DTW(NX, NY) \leq \varepsilon$. Notice that ε is also referred to as a range radius of Y. It is likely that similar subsequences are overlapped, so *Range search* is modified to *Disjoint query*. This means that given all overlapped resultant NXs, *Disjoint query* chooses the one with the smallest $DTW(NX, NY)$.

- *k-nearest neighboring (k-NN) search*: Given a positive integer k, finding a set of k NXs similar to NY, the set is referred to as kS, such that if there is $NX' \notin kS$, then $\forall NX \in kS, DTW(NX, NY) \leq DTW(NX', NY)$. Note that if $k = 1$, the similarity search type becomes *best-so-far search*.

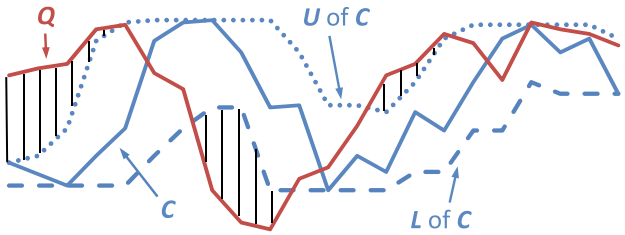

Fig. 6 Reversed LB_Keogh on C and Q

3 Related work

There have been a few typical research efforts dealing with similarity search over streaming time series under DTW. The first is SPRING introduced by Sakurai et al. [13]. The method is very impressive in the computational time. The authors claimed that SPRING is up to 650,000 times faster than using the naive calculation of the DTW distance. However, SPRING cannot work on z-score normalization, since at each time tick the z-score coefficients (mean and standard deviation) change frequently. When the z-score coefficients change, reusing computed results at previous time ticks is virtually impossible to SPRING. Hence, SPRING cannot be used to compare with our methods. Recently, Gong et al. [21] have introduced NSPRING, an extension of SPING supporting z-score normalization. However, since NSPRING computes current data, created from the current z-score coefficients, and then combines these data with the previous data, created from the previous z-score coefficients, in our opinion the method is inaccurate. With an incessant effort, we have recently developed ISPRING [22], an improved variant of SPRING. ISPRING is SPRING equipped with incremental min–max normalization (see Sect. 4.1). We choose min–max normalization rather than z-score normalization for ISPRING, because the min–max coefficients (minimum and maximum values) of evolving subsequences in streaming time series are occasionally changed, whereas the z-score coefficients of the subsequences are almost changed whenever there is a new-coming data point. For the reason, ISPRING using incremental min–max normalization can use current normalized data with previous normalized data to compute the DTW distance between a new-coming time-series subsequence and a specific time-series pattern if the min–max coefficients of the evolving subsequence are not changed. To extract the min–max coefficients of the new-coming subsequence of one streaming time series on the spot, ISPRING uses a monitoring window anchored at the entry of the time-series stream. The experiments in [21] demonstrated that the size of the monitoring window should be the same length as the pattern. We will compare ESUCR-DTW using incremental min–max normalization with ISPRING in Sect. 5.

Next, Rodpongpun et al. [23] proposed a lower bounding function, referred to as *LB_GUN*, under global constraint, uniform scaling, and z-score normalization. *LB_GUN* inherits from LB_Keogh and is expanded to deal with uniform scaling. Since we assume that time-series data are uniformly scaled already in our methods, it is not suitable to compare our methods with *LB_GUN*.

Last but not least, UCR-DTW [11] is a method of similar search in static time series under DTW supporting z-score normalization. The method is of low computational time and high accuracy, so we will be compare SUCR-DTW with UCR-DTW in Sect. 5. UCR-DTW will be reviewed in the following paragraphs.

The spirit of UCR-DTW is using LB_Kim, LB_Keogh, reversed LB_Keogh, and the naive computation of DTW in a cascading fashion. Reversed LB_Keogh is an exchange of the role of query/data in LB_Keogh; this means that query Q is compared with the envelope of time-series sequence C. Reversed LB_Keogh is computed in a *just-in-time* fashion, only if LB_Kim and LB_Keogh fail to prune. Therefore, UCR-DTW costs a negligible time overhead for reversed LB_Keogh to prune off more classical DTW calculations. Figure 6 shows reversed LB_Keogh that contrasts with LB_Keogh illustrated as in Fig. 5. Besides, UCR-DTW uses the Sakoe–Chiba band as a global constraint to support LB_Keogh, reversed LB_Keogh, and the classical DTW calculation for reduction of the running time.

UCR-DTW is briefly described in Algorithm *UCR-DTW* for *best-so-far* search with some notations defined in Table 1. There are some noticeable issues in the algorithm. At first, the lower bounding functions, which are LB_Kim (line 6), LB_Keogh (line 8), reversed LB_Keogh (line 10), and the procedure *DTW* (lines 13 and 15) all use *q.bsf* as an upper bound for early abandoning of these functions. Next, it is likely that the length of T is very large, so it is necessary to read T into many big sections (e.g. 100,000 data points) (line 2). The construction of E_s (line 3) is carried out using the method of Lemire [24] and then E_s is used in reversed LB_Keogh (line 10). In line 4, c extracted from s is stored in a circular buffer whose length is double that of *q.l*. The z-score coefficients of c are got in line 5; and at the moment, the data points of *nc* had not been determined yet. Thanks to these z-score coefficients, the data points of *nc*, which are required in LB_Kim (line 6) and LB_Keogh (line 8), are computed in a *just-in-time* fashion. LB_Kim in line 6 is expanded to cal-

Table 1 Notations for UCR-DTW

Notation	Meaning
T	A static time series
s	A big section of T
q	A time-series pattern for query
nq	The normalized pattern of q
$q.l$	The length of q
$q.r$	The width r of the Sakoe–Chiba band of q
E_s	The envelope of s constructed with $q.r$
$q.bsf$	The *best-so-far* value of q over T
c	A new-coming subsequence of S, corresponding with q
$coef$	The normalization coefficients of c
nc	The normalized pattern of c
$cb1$, $cb2$	Two arrays of cumulative bounds

culate on the first three data points and the last three ones of nq and nc. In line 8, LB_Keogh uses the available envelope of nq. If the algorithm reaches line 10, then the data points of nc are completely determined at the moment, so reversed LB_Keogh does not need to recalculate them. The computation of LB_Keogh on nq and nc in line 8 returns one lower bounding distance between the two normalized time-series sequences, LB_Keogh1, and one array of cumulative bounds, $cb1$. Similarly, in line 10, the calculation of reversed LB_Keogh on nc and nq returns LB_Keogh2 and $cb2$. The procedure *DTW* in lines 13 and 15 uses $cb1$ and $cb2$ for early abandoning of DTW.

Algorithm *UCR-DTW*(T, q)

1. $q.bsf \leftarrow \infty$
2. Get each s of T
3. Construct E_s
4. Use a sliding window whose width is $q.l$ to slide over s. The operation extracts c for every sliding step.
5. $coef \leftarrow$ Incremental z-score normalize c
6. $lb\_Kim \leftarrow LB\_Kim(nq, nc, q.bsf)$
7. **if** $lb\_Kim < q.bsf$ **then**
8. $(lb\_Keogh1, cb1) \leftarrow LB\_Keogh(nq, nc, q.bsf)$
9. **if** $lb\_Keogh1 < q.bsf$ **then**
// reversed LB_Keogh
10. $(lb\_Keogh2, cb2) \leftarrow LB\_Keogh(nc, nq, q.bsf)$
11. **if** $lb\_Keogh2 < q.bsf$ **then**
12. **if** $lb\_Keogh1 > lb\_Keogh2$ **then**
13. $d \leftarrow DTW(nq, nc, q.bsf, cb1)$
14. **else**
15. $d \leftarrow DTW(nc, nq, q.bsf, cb2)$
16. **if** $d < q.bsf$ **then**
17. $q.bsf \leftarrow d$

It is obvious that UCR-DTW combines all the techniques mentioned in Sect. 2.2 so as to accelerate the similarity search; therefore, the method is of low computational time with respect to theoretical evaluation. We also note that the algorithm can be expanded for range search and *k-NN* search, and can be adjusted to process numerous patterns at every sliding step over T.

Since Rakthanmanon et al. [11] reckoned that the speedup techniques used in UCR-DTW dwarf the improvements gained by multi-threading, we want to check the claim and so propose our methods working on multicores. To evaluate the accuracy and execution time of the proposed methods, we adjust the methods for range search and each pattern has its own range radius.

4 The proposed methods

In the section, we present incremental data normalization to support data normalization in the streaming setting and then give the problem definition. After that, the two proposed methods, SUCR-DTW and ESUCR-DTW, are proposed to solve the problem.

4.1 Incremental data normalization

Since time-series sequences change continuously in the streaming setting, data normalization becomes a burden for pre-processing time-series data prior to subsequence matching. Therefore, it is necessary to have a complementary technique for data normalization in the streaming context. We propose incremental data normalization to get normalization coefficients, which are min–max and z-score coefficients, on the fly. Incremental min-max normalization and z-score normalization are presented as follows.

- *Incremental min–max normalization* In the beginning, an ascending numeric array is created from the data points of X with the algorithm of *Quicksort*, so $x_{\min}$ is the first element and $x_{\max}$ is the last one of the ordering array. When there is a new-coming data point, the oldest data point of X is deleted out of the array, and then the new data point is inserted into the array. The course of the deletion and insertion must preserve the ascending order of the array, so the algorithm of *Binary search* is used to find the element that needs deleting and the suitable position in the array to insert the new data point. As *Quicksort* is carried out once when the array of new-coming data points is full at the beginning of the course of the similarity search, and since then *Binary search* is invoked for every new-coming data point afterward, the time complexity of incremental min–max normalization is $\mathcal{O}(\log(n))$.

- *Incremental z-score normalization*

Let us define $x^2 = \sum_{i=1}^{n} x_i^2$. (11)

Equation (10) can be expressed as: $\sigma^2 = \frac{x^2}{m} - \mu^2$. (12)

At first, Eq. (9) is used to compute μ and Eq. (11) is used to compute $x2$. Next, when there is a new-coming data point x_{n+1} deriving from the evolution of X, we compute

$$\mu_{\text{new}} = \mu + \frac{x_{n+1} - x_1}{n} \quad (13)$$

$$\text{and} \quad x_{\text{new}}^2 = x^2 + x_{n+1}^2 - x_1^2. \quad (14)$$

Therefore, we do not need to compute μ_{new} and x_{new}^2 completely. Note that the time complexity of incremental z-score normalization is higher than that of incremental min–max normalization, since the complex arithmetic operators, which are the square to compute x_{new}^2 and the square root to compute σ, are used in incremental z-score normalization.

Notice that because of the accumulation of the floating-point error in the implementation of the incremental z-score normalization, μ_{new} and x_{new}^2 will be completely calculated by Eqs. (9) and (11), respectively, to flush out any accumulated error for once every 100,000 coming data points of a time-series stream.

4.2 Problem definition

The problem is that numerous prespecified time-series patterns need to conduct one of the tasks of similarity search, which is mentioned in Sect. 2.4, over multiple concurrent time-series streams at high-speed rates under DTW and data normalization. The multi-threading technique is proposed to use to support the solution of the problem. The solution consists of two phases:

Phase 1 The patterns are normalized and their envelopes are constructed.

Phase 2 Each threading process deals with one time-series stream. When there is a new-coming data point of the time-series stream, for each pattern the matching procedure will determine whether the new-coming time-series subsequence is a candidate in case of SUCR-DTW or many new-coming time-series subsequences are candidates in case of ESUCR-DTW. The matching procedure works nearly the same as for UCR-DTW. With respect to range search, a similar subsequence has the DTW distance between its normalized subsequence and the normalized pattern within the range radius of the pattern.

Table 2 Additional notations

Notation	Meaning
QSet	The set of patterns
q.ep	The range radius of q
q.RSet	The range set of q
S	A streaming time series
T_n	The new-coming data value at time point n of S
T_{n-1}	The new-coming data value at time point $n-1$ of S
E_c	The envelope of c

There are some following supplementary ideas for the similarity search. As regards data normalization, the similarity search makes a choice between incremental z-score normalization and incremental min–max normalization to get normalization coefficients. Using whichever data normalization is dependent on the requirement of applications. Next, to accelerate the similarity search, data points of one time-series stream are stored in a circular buffer whose length is longer than the length of the longest pattern.

As for one time-series stream, if the similarity search is conducted over one new-coming time-series subsequence of the same length as the pattern, we propose SUCR-DTW, which stands for Streaming UCR-DTW; otherwise, ESUCR-DTW standing for Extended SUCR-DTW is proposed to carry out the similarity search over many new-coming time-series subsequences. Combined with Table 1, Table 2 shows some additional notations, which we will use in the proposed two methods afterwards.

4.3 SUCR-DTW

The working environment of the method is illustrated as in Fig. 7. Given one pattern q, SUCR-DTW conducts the similarity search over the new-coming subsequence c of the time-series stream S and c having length of $q.l$. Therefore, the method has a delay of $q.l$ time ticks at the beginning

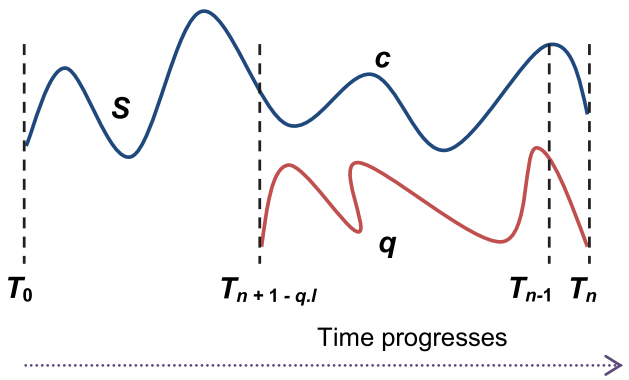

Fig. 7 SUCR-DTW extracts c and normalizes it, and then compares nc with nq under DTW

of the similarity search for q. In addition, when there is a new-coming data point of the time-series stream, the method needs to construct the envelope of c, which is E_c. Notice that q also has its own envelope, which is created in Phase 1, and the data structure to store the envelope is a pair of conventional arrays. In UCR-DTW, the construction of the envelope of c is carried out once for each big section of static time series (line 3 of Algorithm *UCR-DTW*), while in the streaming context, E_c can be created in a *just-in-time* fashion; that is, E_c can be created from scratch, right from the moment at which it is needed. However, SUCR-DTW uses another way to construct E_c, which performs incremental updates on E_c. SUCR-DTW thus uses two circular buffers of the same length as the pattern to store the upper and lower bounds of E_c. In Sect. 5.1, we will compare the performance of the two cases in constructing E_c.

SUCR-DTW implementing Phase 2 is presented in Algorithm *SUCR-DTW* for range search. We have some remarks on the algorithm. Firstly, the algorithm handles many patterns in succession (line 1), and every pattern q has its own subsequence c of the streaming time series S (line 3). Next, construction of E_c is carried out once (line 5), and after that E_c is incrementally updated (line 7). E_c is used in reversed LB_Keogh (line 13). Line 20 indicates that if c is a disjoint query in the set $q.RSet$, then c is a valid similar subsequence of q. Similar to the role of $q.bsf$ in Algorithm *UCR-DTW*, $q.ep$ is an upper bound for early abandoning in LB_Kim (line 9), LB_Keogh (line 11), reversed LB_Keogh (line 13), and the procedure *DTW* (lines 16 and 18).

Algorithm *SUCR-DTW*(S)

When there is a new-coming data of S, T_n

1. **foreach**(q **in** $QSet$)
2. **if** $n \geq q.l - 1$ **then**
3. Get c
4. **if** $n = q.l - 1$ **then**
5. Construct E_c
6. **else**
7. Incrementally update E_c
8. $coef \leftarrow$ Incrementally normalize c
9. $lb\_Kim \leftarrow LB\_Kim(nq, nc, q.ep)$
10. **if** $lb\_Kim \leq q.ep$ **then**
11. $(lb\_Keogh1, cb1) \leftarrow LB\_Keogh(nq, nc, q.ep)$
12. **if** $lb\_Keogh1 \leq q.ep$ **then**
// reversed LB_Keogh
13. $(lb\_Keogh2, cb2) \leftarrow LB\_Keogh(nc, nq, q.ep)$
14. **if** $lb\_Keogh2 \leq q.ep$ **then**
15. **if** $lb\_Keogh1 > lb\_Keogh2$ **then**
16. $d \leftarrow DTW(nq, nc, q.ep, cb1)$
17. **else**
18. $d \leftarrow DTW(nc, nq, q.ep, cb2)$
19. **if** $d \leq q.ep$ **then**
20. **if** $DisjoinQuery(c, q.RSet)$ **then**
21. Add c into $q.RSet$
22. **end foreach**

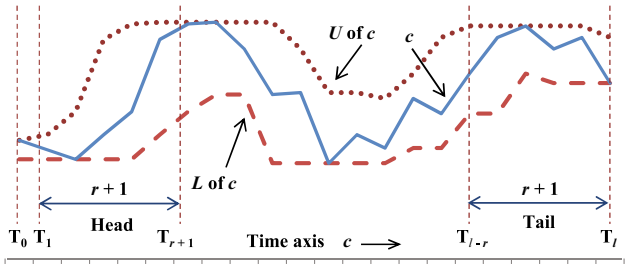

Fig. 8 The parts of *Head* and *Tail* of $E_c = \{U, L\}$ need updating

Next, we will depict how to update E_c incrementally. When a data point of S newly arrives, we can imagine that c slides rightwards along the time axis. Let l be the length of c, and r be the width of the Sakoe–Chiba band of q ($r \ll l$). In other words, $q.l$ is referred to as l, and $q.r$ is referred to as r. For the sake of illustration, we assume that c slides from time tick 0 to 1, and l is the most recent time tick. That means $c = \{T_1, T_2, \ldots, T_l\}$. Let $U = \{u_1, u_2, \ldots, u_l\}$ be the upper bound of c, and $L = \{l_1, l_2, \ldots, l_l\}$ be the lower bound of c. That means $E_c = \{U, L\}$. Deduced from the formula (5), U and L keep the same values from time points $r + 2$ to $l - r - 1$. Let *Head* be values of E_c from time point 1 to $r + 1$ and *Tail* be values of E_c from time point $l - r$ to l. *Head* and *Tail* need updating when c slides over the time axis. Figure 8 depicts the envelope E_c, and its *Head* and *Tail* when c slides from time tick 0 to 1.

It is obvious that if E_c is completely constructed for each sliding step, the time complexity of the task is $\mathcal{O}(l)$; however, if E_c is incrementally updated at its *Head* and *Tail*, the time complexity reduces to $\mathcal{O}(r)$. Furthermore, due to the features of the upper and lower bounds of E_c, updating *Head* and *Tail* can be early abandoned. Updates on *Head* and *Tail* are detailed in Algorithm *UpdateTail* and Algorithm *UpdateHead*, respectively. Line 7 of Algorithm *SUCR-DTW* implements *UpdateTail* as well as *UpdateHead*.

Algorithm *UpdateTail*

1. $u_l \leftarrow \max(T_{l-r}, T_{l-r+1}, \ldots, T_l)$ // update U
2. **for** $(i = 1; i \leq r; i{+}{+})$
3. **if** $u_{l-i} \geq u_l$ **then**
4. **break**
5. $u_{l-i} \leftarrow u_l$
6. **end for**
7. $l_l \leftarrow \min(T_{l-r}, T_{l-r+1}, \ldots, T_l)$ // update L
8. **for** $(i = 1; i \leq r; i{+}{+})$
9. **if** $l_{l-i} \leq l_l$ **then**
10. **break**
11. $l_{l-i} \leftarrow l_l$
12. **end for**

Algorithm *UpdateHead*

1. **if** $T_0 = u_0$ **then** // update U
2. $u_1 \leftarrow \max(T_1, T_2, \ldots, T_{r+1})$
3. $umax \leftarrow u_1$
4. **for** $(i = 2; i \leq r + 1; i{+}{+})$
5. **if** $T_{r+i} > umax$ **then**
6. $umax \leftarrow T_{r+i}$
7. $u_i \leftarrow umax$
8. **end for**
9. **if** $T_0 = l_0$ **then** // update L
10. $l_1 \leftarrow \min(T_1, T_2, \ldots, T_{r+1})$
11. $lmin \leftarrow l_1$
12. **for** $(i = 2; i \leq r + 1; i{+}{+})$
13. **if** $T_{r+i} < lmin$ **then**
14. $lmin \leftarrow T_{r+i}$
15. $l_i \leftarrow lmin$
16. **end for**

Note that if the condition in line 3 of Algorithm *UpdateTail* is satisfied, updating U will be immediately stopped. Similarly, early abandoning takes place for L if the condition in line 9 of the algorithm is true. As regards Algorithm *UpdateHead*, we note that $u_0 = \max(T_0, T_1, \ldots, T_r)$ and $l_0 = \min(T_0, T_1, \ldots, T_r)$ for the present. Updating U in *Head* occurs only if T_0 is u_0 (line 1). Similarly, updating L in *Head* takes place only if T_0 is equal to l_0 (line 9). Notice that *UpdateTail* and *UpdateHead* need adjusting slightly in the general case where c slides from time tick h to $h + 1$ with $h \geq 0$.

To evaluate the effectiveness and efficiency of the incremental updates on E_c, we make two different variants of SUCR-DTW. Both of them do not have lines 4–7 of Algorithm *SUCR-DTW*. The first variant creates E_c at the right time at which E_c is needed (between lines 12 and 13 of Algorithm *SUCR-DTW*). Let us denote the variant by SUCR-DTW-1. The second variant does not use E_c, so reversed LB_Keogh is omitted (lines 13, 14, 15, 17, and 18 are ignored). Let us denote the second variant by SUCR-DTW-2. Besides, we also modify UCR-DTW so that the algorithm can accommodate with multi-threaded programming, in which each threading process handles a static time-series sequence. Let TUCR-DTW be UCR-DTW equipped with multi-threading.

4.4 ESUCR-DTW

The method is an extension of SUCR-DTW from the following observation. Given a pattern q, the subsequences at the entry of one streaming time series can be matched with the pattern under DTW provided that the lengths of these subsequences are within a range of $[q.l - q.r : q.l + q.r]$. The reason is that because of the characteristic of the Sakoe–Chiba band, LB_Keogh, reversed LB_Keogh, and the procedure *DTW* can work with q and a time-series subsequence whose length is within this range. Figure 9 indicates

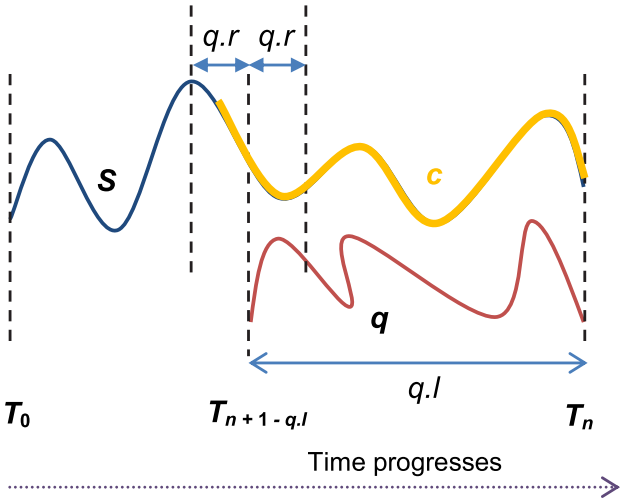

Fig. 9 A new-coming subsequence c whose length is within a range of $[q.l - q.r : q.l + q.r]$ can be matched with pattern q by ESUCR-DTW

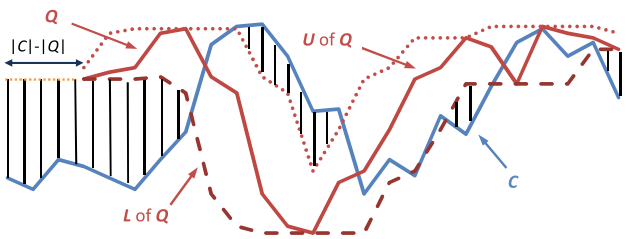

Fig. 10 Using LB_Keogh in case of $0 \leq |C| - |Q| \leq r$

that a subsequence c can be matched with q as the length of c is within the range. ESUCR-DTW can find many more similar subsequences than SUCR-DTW does, because the latter performs similarity search only over one new-coming subsequence of length $q.l$. The trade-off of finding many more similar subsequences by ESUCR-DTW is that the running time of ESUCR-DTW is much longer than that of SUCR-DTW, directly proportional to the number of subsequences matched with q at a time tick. The maximum number of subsequences that are performed by the similarity search for every pattern q over one time series stream at a time tick is $2 \times q.r + 1$.

Original LB_Keogh works only on two time-series sequences of the same length, so to deal with two time-series sequences whose lengths are different within a width r of the Sakoe–Chiba band, the lower bounding function needs a slight change in the two following cases.

Figure 10 illustrates the first case in which C is longer than Q. Let $m = |Q|, n = |C|$, and $w = n - m$. Suppose that $w \leq r$. It is intuitive that $c_1, c_2, \ldots, c_w$ can be measured with the first couple of the envelope of Q (i.e. u_1 and l_1). The formula (6) is thus changed to

$$LB\_Keogh(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (c_i - u_1)^2 & \text{if } i \le w \text{ and } c_i > u_1 \\ (l_1 - c_i)^2 & \text{if } i \le w \text{ and } c_i < l_1 \\ (c_i - u_{i+1-w})^2 & \text{if } i > w \text{ and } c_i > u_{i+1-w} \\ (l_{i+1-w} - c_i)^2 & \text{if } i > w \text{ and } c_i < l_{i+1-w} \\ 0 & \text{otherwise.} \end{cases}} \quad (15)$$

Figure 11 shows the second case in which Q is longer than C, which means $w < 0$. The formula (6) becomes

$$LB\_Keogh(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (c_i - u_{i+1-w})^2 & \text{if } i > w \text{ and } c_i > u_{i+1-w} \\ (l_{i+1-w} - c_i)^2 & \text{if } i > w \text{ and } c_i < l_{i+1-w} \\ 0 & \text{otherwise.} \end{cases}} \quad (16)$$

Let $q.cS$ denote the set of new-coming subsequences, which are performed the similarity search for q, of streaming time series S. Also, given $\alpha, \beta \in N$ and $\alpha, \beta \le q.r$, all the subsequences of $q.cS$ have their lengths within a range of $[q.l - \alpha : q.l + \beta]$. ESUCR-DTW implementing Phase 2 is described in Algorithm *ESUCR-DTW*. There are some comments on the algorithm as follows. Line 2 implies that the similarity search has a delay of $q.l - q.r$ time ticks. At the time tick $q.l - q.r - 1$, the envelope E_c of a new-coming subsequence of length $q.l - q.r$ is created (line 4). This envelope is used in reserved LB_Keogh for all subsequences in $q.cS$. Notice that E_c does not have its *Head* at this moment (line 4), so in the following time ticks from $q.l - q.r$ to $q.l + q.r - 1$, only *Tail* of E_c is updated (line 6). At time tick $q.l + q.r - 1$, E_c is fully made; this means E_c has *Tail* as well as *Head*. Thus, *Head* of E_c is also updated since time tick $q.l + q.r$ (line 8). Line 9 gets all new-coming subsequences of $q.cS$. The similarity search is then performed on every subsequence c in $q.cS$ for pattern q in the same course of the similarity search in Algorithm *SUCR-DTW* (line 11). Note that the maximum number of the elements in $q.cS$ is $2 \times q.r + 1$ and, if $(\alpha, \beta) = (0, 0)$, then ESUCR-DTW becomes SUCR-DTW.

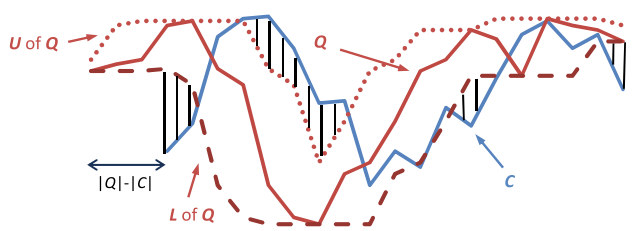

Fig. 11 Using LB_Keogh in case of $0 \le |Q| - |C| \le r$

Algorithm *ESUCR-DTW*(*S*)
When there is a new-coming data of *S*, T_n
1. **foreach**(q **in** $QSet$)
2. **if** $n \ge q.l - q.r - 1$ **then**
3. **if** $n = q.l - q.r - 1$ **then**
4. Construct E_c
5. **else**
6. $UpdateTail$ for E_c
7. **if** $n > q.l + q.r - 1$ **then**
8. $UpdateHead$ for E_c
9. Get $q.cS$
10. **foreach**(c **in** $q.cS$)
11. Reuse lines 8-21 of Algorithm *SUCR-DTW*
12. **end foreach**
13. **end foreach**

5 Experimental evaluation

The section demonstrates experiments on the methods of similarity search to evaluate their effectiveness and efficiency. All the experiments were conducted on an Intel Dual Core i3 M350 2.27 GHz, 4GB RAM PC. The programming language is C# as the language is powerful for multi-threading. For the sake of fairness, all threading processes are of the same priority.

5.1 Evaluation of SUCR-DTW

Table 3 presents Dataset 1 that consists of five time-series text files used as input for five time-series sequences. The sources of the time-series files are given in [25,26]. Except for the first time-series file, the four remaining time-series files are for time-series classification and clustering, but not time-series subsequences matching. Thus, we revised the data format of these four time-series files slightly as follows. Each time-series file consists of a set of time-series sequences $\{S_1, S_2, \ldots, S_n\}$ and label l_i(s) of the sequences. We connect all the sequences $S_1, S_2, \ldots, S_n$ together as a whole sequence S. That means the labels l_i(s) in the time-series file are removed.

We created three pattern sets from the above time-series files, and the number of patterns in each pattern set is 100. In

Table 3 Dataset 1 simulates time-series sequences

No.	Time-series file	Length
1	Data.txt to demonstrate UCR Suite	1,000,000
2	Revised CinC_ECG_torso_TEST	2,261,820
3	Revised InlineSkate_TEST	1,035,100
4	Revised NonInvasiveFatalECG_Thorax1TEST	1,271,250
5	Revised uWaveGestureLibrary_X_TEST	1,128,330
	Total points	6,695,500

each pattern set, the number of patterns created from a time-series file is directly proportional to the number of data points in the file. The lengths of the patterns for query vary from 50 to 550. The patterns were extracted from random positions in the time-series files. Next, all data points of a pattern were added by a numerical constant, and then the data points were virtually increased or decreased by a relatively small numeric value (e.g. 0.3 or −0.3). Finally, 33 % of the data points were changed by which they got the value of the preceding data point or successive one, or the mean of neighboring ones. The total number of data points was 28,762 for the pattern set 1; 28,927 for the pattern set 2; and 33,219 for the pattern set 3.

We developed UCR-DTW for *best-so-far* search as well as range search for numerous prespecified patterns in multiple static time series. After that, we implemented UCR-DTW for *best-so-far* search of each pattern set over Dataset 1 and recorded the *best-so-far* value of every pattern. The *best-so-far* value is used as a range radius of this pattern for range search. In this way, we created three pattern sets for range search over Dataset 1. The other parameters of the testbed are as follows. The circular buffers of the time-series streams have the size of 1024. Since the authors in [7] claimed that 10 % constraint on warping inherited from speech community is actually much higher than the constraint needed for data-mining applications, the methods of similarity search takes 5 % constraint on warping in their implementation. For example, if the length of a time-series pattern is 500, then the width r of the Sakoe–Chiba band is 25.

Then we implemented UCR-DTW for range search of the three pattern sets. The results are 100 hits for each pattern set, and all the patterns have one hit. After that, we carried out SUCR-DTW, SUCR-DTW-1, SUCR-DTW-2, and TUCR-DTW for the three pattern sets. Notice that all the methods in the experiments use incremental z-score normalization mentioned in Sect. 4.1. The results of the methods are same as those of UCR-DTW, so the precision and recall of the methods are 100 %. As regards wall clock time, the obtained results are illustrated as in Fig. 12. The figure shows that the performance of SUCR-DTW is better than those of SUCR-DTW-1 and SUCR-DTW-2. This means that incrementally updating the envelope E_c of a time-series subsequence c and then using E_c in reversed LB_Keogh make similarity search faster than completely constructing E_c, or not using E_c. UCR-DTW is slowest; however, TUCR-DTW, which is UCR-DTW equipped with multi-threading, has the least wall clock time. Note that TUCR-DTW is not suitable for the streaming setting, because the method requires available time-series sequences before the similarity search is done, whereas time-series data points of a time-series stream are only collected at every time tick, unknown beforehand. Finally, with regard to SUCR-DTW, we recorded the average CPU times to process a new-coming data point of 2835 ticks for the pattern set 1; 2944 ticks for the pattern set 2; and 3208 ticks for the pattern set 3. These results present the usability of SUCR-DTW in real-time applications that need to perform the similarity search at high-speed rates.

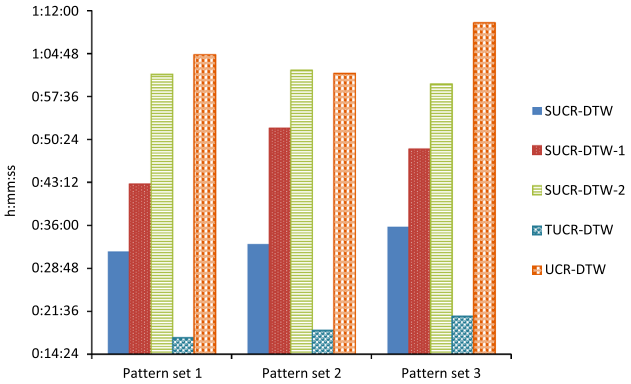

Fig. 12 Statistic of the wall clock times for range search

Table 4 Pruning powers of the lower bounding functions

	LB_Kim (%)	LB_Keogh (%)	Reversed LB_Keogh (%)
Pattern set 1			
SUCR-DTW	54.66	34.67	9.95
UCR-DTW	54.67	34.66	9.90
Pattern set 2			
SUCR-DTW	55.45	34.41	9.44
UCR-DTW	55.45	34.40	9.39
Pattern set 3			
SUCR-DTW	51.47	39.17	8.69
UCR-DTW	51.47	39.17	8.65

Finally, there are two noticeable remarks on pruning powers of the three lower bounding functions used in SUCR-DTW and UCR-DTW. In the two methods, LB_Kim, LB_Keogh, and reversed LB_Keogh are arranged in the ascending tightness of the lower bounding property, such that front lower bounding functions with low time complexities rule out most unpromising subsequences. As a result, the number of post-checking times using the classical DTW so as to determine if a candidate subsequence is a true hit is very tiny. Table 4 indicates that LB_Kim, whose time complexity is $\mathcal{O}(1)$, takes charge of the most pruning. After that, LB_Keogh, whose time complexity is $\mathcal{O}(n)$, prunes off remaining unpromising subsequences, and then reversed LB_Keogh, whose time complexity is highest, tries to prune off unpromising subsequences which LB_Kim and LB_Keogh cannot detect. In addition, Table 4 shows that the pruning power of reversed LB_Keogh in SUCR-DTW is larger than that in UCR-DTW, roughly 0.05 %. This implies that the envelopes, which are incrementally updated in SUCR-DTW, are tighter than those which are constructed once, in UCR-

Table 5 Dataset 2 simulates time-series sequences

No.	Time-series file	Length
1	Revised Adiac_TEST	68,816
2	Revised Adiac_TRAIN	68,640
3	Revised FISH_TEST	81,025
4	Revised FISH_TRAIN	81,025
5	Revised MedicalImages_TEST	75,240
	Total points	374,746

Table 6 The number of same similar subsequences of the two incremental data normalization

(α, β)	# of same similar subsequences
(1, 1)	8
(2, 2)	6
(3, 3)	9
(4, 4)	4
(5, 5)	2

DTW. In SUCR-DTW, more tightness is shown at both ends of the envelopes of new-coming subsequences.

5.2 Evaluation of ESUCR-DTW

To evaluate ESUCR-DTW, we used Dataset 2 depicted as in Table 5. The dataset consists of five time-series files [25] revised in the same way as the four time-series files done in Dataset 1. The reason using Dataset 2 rather than Dataset 1 is that the time complexity of ESUCR-DTW is much larger than that of SUCR-DTW, so experiments on ESUCR-DTW with Dataset 1 will have wall clock times too long. Although the size of Dataset 2 is smaller than that of Dataset 1, the experimental results on Dataset 2 are also significant, because the second dataset has a comparatively large scale.

Next, a pattern set was created from the revised time-series files of Dataset 2. The number of patterns in the set is 100 and the lengths of the patterns vary from 128 to 512. These patterns were created in the same way as the patterns constructed in Sect. 5.1. The total number of data points of the pattern set is 34,404. Three next experiments on Dataset 2 and the pattern set were carried out to evaluate ESUCR-DTW.

At first, ESUCR-DTW was compared with SUCR-DTW in terms of accuracy and wall clock time. In the experiment, both the methods use incremental z-score normalization. As regards ESUCR-DTW, we set $(\alpha, \beta) = (1, 1)$. At the beginning of the comparison, we implemented an original UCR-DTW on the pattern set and recorded the *best-so-far* value of every pattern. The *best-so-far* value is used as a range radius of the pattern for range search. We thus created the pattern set for range search over Dataset 2. After that, SUCR-DTW and ESUCR-DTW were implemented. With regard to SUCR-DTW, the results are 100 hits, and all patterns have 1 hit. The wall clock time of SUCR-DTW is roughly 8:56.95 min. With respect to ESUCR-DTW, the results are 111 hits, and the patterns have one or many hits. However, the wall clock time of ESUCR-DTW is about 26:28.02 min. In the experiment, ESUCR-DTW is three times as slow as SUCR-DTW, since for each pattern q, ESUCR-DTW must perform similarity search over three new-coming subsequences, whose lengths are $q.l - 1$, $q.l$, and $q.l + 1$, respectively, of one streaming time series at a time tick. It is obvious that ESUCR-DTW is more accurate than SUCR-DTW, yet the former costs much more time than the latter.

The second experiment illustrates the correlation between the obtained results with incremental z-score normalization and those with incremental min-max normalization. At first, ESUCR-DTW is changed from range search to *best-so-far* search, and then the method employs incremental z-score normalization with the cases of 2-tuple (α, β) be (0, 0), (1, 1), (2, 2), (3, 3), (4, 4), and (5, 5). After that, the method uses incremental min–max normalization with the same above cases of (α, β). Table 6 presents the number of same *best-so-far* subsequences obtained by ESUCR-DTW using incremental z-score normalization and ESUCR-DTW using incremental min–max normalization for each case of (α, β). These results show that if the search scope of new-coming subsequences increases in ESUCR-DTW (i.e. from (1, 1) up to (5, 5)), the number of same similar subsequences of the two cases tends to decrease. As a whole, the *best-so-far* subsequences obtained with incremental z-score normalization rarely coincide with those done with incremental min–max normalization. Notice that in case of *best-so-far* search implemented with SUCR-DTW or ESUCR-DTW, a threading process can compete with others to update the *best-so-far* value of one pattern at a time, so the system must lock the shared property and check the *best-so-far* value again before the update can be done.

Other fascinating statistics of the second experiment also is unfolded. We reuse the conventional symbols as follows. Let q be a time-series pattern for query and c be the *best-so-far* subsequence of q. Figure 13 depicts the number of cases where *best-so-far* subsequences are shorter than patterns ($|c| < |q|$), longer than patterns ($|c| > |q|$), and the same length as patterns ($|c| = |q|$) for each case of (α, β). ESUCR-DTW uses incremental z-score normalization and incremental min–max normalization in turn. The figure reveals that ESUCR-DTW often returns similar subsequences longer than patterns. With regard to both incremental data normalizations, the number of cases of $|c| = |q|$ is relatively low; especially for incremental z-score normalization, the value is from four to five cases. Next, the number of

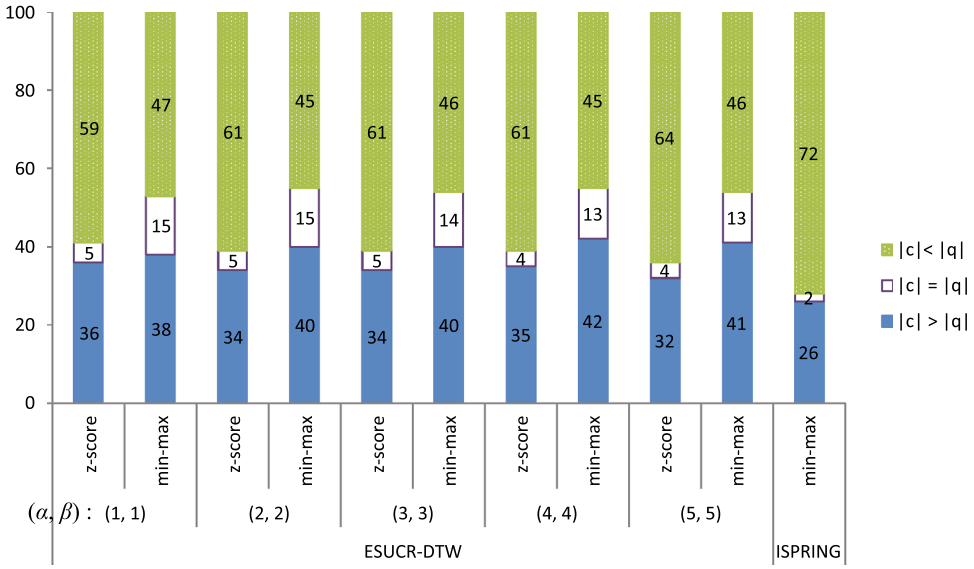

Fig. 13 Evaluate *best-so-far* subsequences with their patterns in terms of length in 100 cases

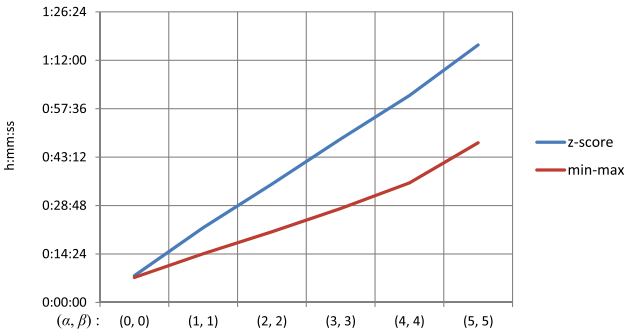

Fig. 14 The wall clock times of ESUCR-DTW for each case of (α, β) with the two incremental data normalizations

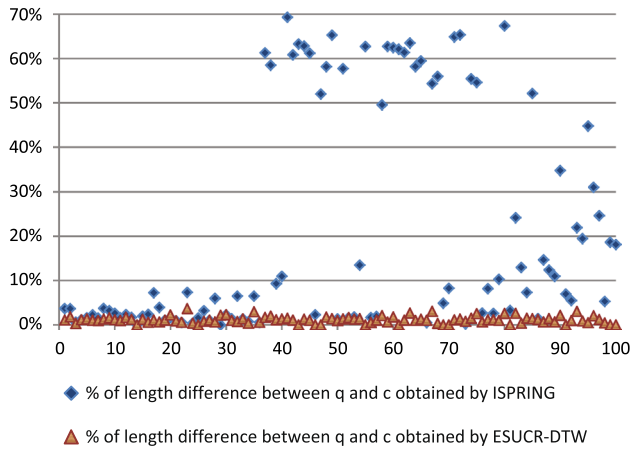

Fig. 15 The distribution of the percentages of length difference between the 100 patterns and the 100 corresponding *best-so-far* subsequences obtained by ISPRING and ESUCR-DTW

cases of $|c| > |q|$ with incremental min–max normalization is larger than that with incremental z-score normalization.

Figure 14 indicates the wall clock times of ESUCR-DTW for every case of (α, β) with the two incremental data normalizations. Note that if ESUCR-DTW is degraded to SUCR-DTW due to $(\alpha, \beta) = (0, 0)$, the wall clock time of ESUCR-DTW using incremental z-score normalization and that of ESUCR-DTW using incremental min–max normalization are nearly the same. Figure 14 also depicts that the wall clock times of ESUCR-DTW using incremental min–max normalization are smaller than those of ESUCR-DTW using incremental z-score normalization. However, this does not mean that ESUCR-DTW supports incremental min–max normalization better than incremental z-score normalization. Incremental min–max normalization takes low computational time evidently, $\mathcal{O}((\log(n)))$, so the technique mitigates the running time of ESUCR-DTW significantly; whereas incremental z-score normalization is of higher computational time, ESUCR-DTW has longer wall clock time. Furthermore, the wall clock times of ESUCR-DTW using incremental z-score normalization are nearly directly proportional to the number of new-coming subsequences of one streaming time series, which are matched with a pattern at a time tick. For instance, the wall clock times are 7:57.41 min in case of $(\alpha, \beta) = (0, 0)$, and 1:16:33.94 h in case of $(\alpha, \beta) = (5, 5)$. The increase of approximately ten times in wall clock time relates to the increase in the number of new-coming subsequences on which the similarity search is performed with a specific pattern at a time tick, of one time-series stream, from 1 up to $5 + 1 + 5 = 11$.

The third experiment is to compare ESUCR-DTW and ISPRING in terms of wall clock time and the quality of similar subsequences. *Best-so-far* search using min–max normalization is implemented in the two methods. We use again results obtained by ESUCR-DTW with $(\alpha, \beta) = (5, 5)$. The wall clock times of ESUCR-DTW and ISPRING are 47:27.75 min, 1:37:13.88 h, respectively. Thus, ESUCR-DTW is roughly twice as fast as ISPRING in the testbed.

With respect to similar subsequences, there are 91 cases in which *best-so-far* values obtained by ISPRING are smaller (i.e. better) than those done by ESUCR-DTW. The two methods have the same *best-so-far* values in the nine remaining cases. As regards ISPRING, there are two remarks on lengths of similar subsequences. Firstly, Fig. 13 indicates that the probability by which ISPRING returns a similar subsequence whose length is shorter than that of the corresponding pattern is very high, 72 % of ISPRING compared with 46 % of ESUCR-DTW. Secondly, the patterns and their similar subsequences often have a huge difference of lengths. For example, there is a pattern of length 479, and ISPRING returns its *best-so-far* subsequence of length 186; the percentage of length difference in this case is 61.17 %. ESUCR-DTW returns similar subsequences whose lengths are dependent on (α, β), so the similar subsequences always have lengths in a domain prespecified by users. Figure 15 shows that with respect to ISPRING, the maximum percentage of length difference is 69.3 %, whereas the value is 3.68 % as for ESUCR-DTW. If the length of a similar subsequence is too different from that of the corresponding pattern, it is because ISPRING does not use any constraint on the warping path in the accumulated cost matrix. It is likely that a point of one pattern matches with too many points of its similar subsequence, or vice versa. Consequently, ISPRING can create a pathological warping path in the accumulated cost matrix and this is a shortcoming of this method.

6 Conclusions and future work

The paper has presented two methods of similar search for numerous prespecified patterns in multiple time-series streams under DTW. The both methods conduct data normalization before the DTW distance between two normalized time-series sequences is computed. To be adaptive in the streaming setting, these methods use either incremental z-score normalization or incremental min–max normalization. The first method, SUCR-DTW [15], is a modification of UCR-DTW, a state-of-the-art method of similar search for prespecified patterns in static time series, so that SUCR-DTW can cope with difficulties and complexities of similarity search in streaming time series. Furthermore, SUCR-DTW can deal with multiple concurrent time-series streams at high-speed rates, because the method employs multi-threading and a combination of techniques so as to accelerate the performance of the similarity search. One of these techniques is that the envelopes of new-coming time-series subsequences are incrementally updated with low computational time, so these envelopes can be immediately used in reversed LB_Keogh, a lower bounding function to prune off more unpromising subsequences. However, like UCR-DTW, SUCR-DTW returns similar subsequences of the same length as the patterns. We thus introduce the second method, ESUCR-DT, which is an extension of SUCR-DTW for finding similar subsequences whose lengths are likely different from those of the patterns. Some major conclusions are drawn from the experiments on SUCR-DTW, ESUCR-DTW, and ISPRING, and another method of similar search over time-series streams, as follows.

- SUCR-DTW has the same precision as UCR-DTW and runs faster than the original UCR-DTW without multi-threading.
- The envelopes incrementally updated in SUCR-DTW are tighter than those constructed once in UCR-DTW.
- Similar subsequences obtained by ESUCR-DTW are better than those done by SUCR-DTW, yet the former must spend more time than the latter.
- With regard to ESUCR-DTW, the *best-so-far* subsequences obtained with incremental z-score normalization rarely coincide with those done with incremental min–max normalization.
- ESUCR-DTW often returns similar subsequences longer than the patterns.
- In addition, ESUCR-DTW using incremental z-score normalization is slower than ESUCR-DTW using incremental min–max normalization.
- *Best-so-far* values obtained by ISPRING are less than or equal to those done by ESUCR-DTW, yet ISPRING often returns *best-so-far* subsequences whose lengths are unreasonable. This means that many similar subsequences obtained by ISPRING are too short in comparison to the lengths of the patterns.

In future work, we plan to study how to identify common local patterns of coevolving time-series sequences under DTW and data normalization in light of the outcomes obtained from this work.

Acknowledgments We specially thank the authors in [11] for providing the source code and datasets of the UCR suite on the website [26]. Thanks to the valuable resources, our work was quickly progressed.

References

1. Borgne, Y.-A., Santinib, S., Bontempi, G.: Adaptive model selection for time series prediction in wireless sensor networks. Signal Process. **87**(12), 3010–3020 (2007)

2. Zhu, Y., Shasha, D.: Efficient elastic burst detection in data streams. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, pp. 181–192 (2003)
3. Wu, H., Salzberg, B., Zhang, D.: Online event driven subsequence matching over financial data streams. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, New York, USA, pp. 23–34 (2004)
4. Yang, Q., Wu, X.: 10 challenging problems in data mining research. Int. J. Inf. Technol. Decis. Mak. 5(4), 597–604 (2006)
5. Fu, T.-C.: A review on time series data mining. J. Eng. Appl. Artif. Intell. **24**, 164–181 (2011)
6. Bemdt, D., Clifford, J.: Using Dynamic Time Warping to find patterns in time series. In: Proceedings of AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, USA, pp. 359–370 (1994)
7. Ratanamahatana, C., Keogh, E.: Everything you know about Dynamic Time Warping is wrong. In: 3rd Workshop on Mining Temporal and Sequential Data, in Conjunction with 10th ACM SIGKDD International Conference Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA, USA (2004)
8. Petitjean, F., Forestier, G., Webb, G., Nicholson, A., Chen, Y., Keogh, E.: Dynamic Time Warping averaging of time series allows faster and more accurate classification. In: ICDM 2014: IEEE International Conference on Data Mining, Shenzhen, China, pp. 470–479 (2014)
9. Zakaria, J., Mueen, A., Eamonn Keogh, E.: Clustering time series using unsupervised-shapelets. In: ICDM 2012: IEEE International Conference on Data Mining, Brussels, Belgium, pp. 785–794 (2012)
10. Keogh, E., Ratanamahatana, C.: Exact indexing of Dynamic Time Warping. Knowl. Inf. Syst. **7**(3), 358–386 (2004)
11. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under Dynamic Time Warping. In: The 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'12), Beijing, China, pp. 262–270 (2012)
12. West, M.: http://www.isds.duke.edu/mw/data-sets/ts_data/. Accessed Dec 2013
13. Sakurai, Y., Faloutsos, C., Yamamuro, M.: Stream monitoring under the time warping distance. In: The IEEE 23rd International Conference on Data Engineering, Istanbul, Turkey, pp. 1046–1055 (2007)
14. Capitani, P., Ciaccia, P.: Warping the time on data streams. Data Knowl. Eng. **62**(3), 438–458 (2007)
15. Giao, B., Anh, D.: Similarity search in multiple high speed time series streams under Dynamic Time Warping. In: Proceedings of 2015 2nd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS 2015), Ho Chi Minh City, Vietnam, pp. 82–87 (2015)
16. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoust. Speech Signal Process. **26**(1), 43–49 (1978)
17. Kim, S.-W., Park, S.: An index-based approach for similarity search supporting time warping in large sequence databases. In: Proceedings of the 17th IEEE International Conference on Data Engineering, Heidelberg, Germany, pp. 607–614 (2001)
18. Junkui, L., Yuanzhen, W.: Early abandon to accelerate exact Dynamic Time Warping. Int. Arab J. Inf. Technol. **6**(2), 144–152 (2009)
19. Tan, S., Lau, P., Yu, X.: Finding similar time series in sales transaction data. In: Proceedings of 28th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2015, Seoul, South Korea, pp. 645–654 (2015)
20. Vinh, V., Anh, D.: Constraint-based MDL principle for semi-supervised classification of time series. In: Proceedings of 2015 Seventh International Conference on Knowledge and Systems Engineering, Ho Chi Minh City, Vietnam, pp. 43–48 (2015)
21. Gong, X., Fong, S., Chan, J., Mohammed, S.: NSPRING: the SPRING extension for subsequence matching of time series supporting normalization. J. Supercomput., pp. 1–25 (2015). doi:10.1007/s11227-015-1525-6
22. Giao, B., Anh, D.: Improving SPRING Method in similarity search over time series streams by data normalization. In: Proceedings of 2nd EAI International Conference on Nature of Computation and Communication, Rach Gia, Vietnam (2016). http://ictcc.org/2016/show/program-final
23. Rodpongpun, S., Niennattrakul, V., Ratanamahatana, C.: Efficient subsequence search on streaming data based on time warping distance. Comput. Inf. Technol. **5**(1), 2–8 (2011)
24. Lemire, D.: Faster retrieval with a two-pass Dynamic-Time-Warping lower bound. Pattern Recognit. **42**(9), 2169–2180 (2009)
25. Keogh, E.: In: The UCR classification/clustering page. http://www.cs.ucr.edu/~eamonn/time_series_data/. Accessed Aug 2013
26. Keogh, E.: In: The UCR Suite. http://www.cs.ucr.edu/~eamonn/UCRsuite.html. Accessed Dec 2014